DEPARTMENT OF THE AIR FORCE
HEADQUARTERS ELECTRONIC SYSTEMS CENTER (AFMC)
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731-5000

JAN 29 1996

MEMORANDUM FOR LORAL FEDERAL SYSTEMS
ATTN: RICHARD HUGHES

FROM: ESC/ENS
5 Eglin Street
Bldg. 1704, Rm. 206
Hanscom AFB, MA 01731-2116

SUBJECT: Upgrade to Distribution Statement A

1. The STARS product, CDRL A014-015, "Integrating Cleanroom with Object Oriented Methods for Reliable Software Development", is upgraded to Distribution Statement A effective 29 Jan 96.

2. Please direct any questions you may have to the Jim Henslee at (617) 377-8563.

ROBERT LENCEWICZ
ESC CARDS Program Manager
Software Design Center

DISTRIBUTION STATEMENT UPGRADE

CDRL Number and Task Number: CDRL A014-015, Task IV02

Product Title and Brief Description (what it is and what it does): "Integrating Cleanroom with Object Oriented Methods for Reliable Software Development"

The purpose of this paper is to 1) discuss why object-oriented and Cleanroom software engineering techniques should be integrated, 2) outline the generic process for object-oriented software development that was derived on STARS Task IA09, and comment on relevant aspects of the mapping from the studied methods to each generic process activity, and 3) discuss the shared leveraging of Cleanroom and object-oriented techniques, and how the integration of these techniques might be leveraged to produce software of greater reliability and reusability.

Date Delivered to the Program Office: 24 Jan 96

Reviewer's Name, Extension Number and Date of Review: Marcelle Nachef, (617) 377-4918, 29 Jan 96

Intended Audience: Public conferences, trade shows, workshops, and STC '96

Comments:

The STARS product, Integrating Cleanroom with Object Oriented Methods for Reliable Software Development, previously under Distribution Statement C, is upgraded to Distribution Statement A effective 29 Jan 96. This product is generic and does not apply to specific defense articles and defense services. In accordance with Memorandum of Agreement between ESC/PA and ESC/ENS concerning upgrades of STARS products to Distribution A, the STARS program office at ESC/ENS has reviewed this product and has determined that the information is unclassified, technically accurate, and suitable for public release.

Approving Authority

1/30/96
Date

Task/Subtask IV02.1
CDRL Sequence A014-015
23 January 1996

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Technical Papers:
Integrating Cleanroom with Object Oriented Methods For Reliable Software Development

Contract No. F19628-93-C-0129

Task IV01 - Megaprogramming Transition Support

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2116

Prepared by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

Task/Subtask IV02.1
CDRL Sequence A014-015
23 January 1996

SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS (STARS) PROGRAM

Technical Papers:
Integrating Cleanroom with Object Oriented Methods For Reliable Software Development

Contract No. F19628-93-C-0129

Task IV01 - Megaprogramming Transition Support

Prepared for:

Electronic Systems Center
Air Force Materiel Command, USAF
Hanscom AFB, MA 01731-2116

Prepared by:

Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

REPORT DOCUMENTION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1/23/96	Initial

4. TITLE AND SUBTITLE
Integrating Cleanroom with OO Methods for Reliable Software Development

5. FUNDING NUMBERS
F19628-93-C-0129

6. AUTHOR(S)
William H. Ett, Loral Federal Systems

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879

8. PERFORMING ORGANIZATION REPORT NUMBER
A014-015

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Electronic Systems Center/ENS
Air Force Materiel Command, USAF
5 Eglin Street, Building 1704
Hanscom Air Force Base, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
N/A

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Cleared for Public Release, Distribution is Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

STARS Task IA09 was conceived to examine the potential complementary nature of the Cleanroom Engineering of software and a representative set of the popular object-oriented methods being used to specify, design and develop software systems. Both methods of softrware development support the software concepts of abstraction, encapsulation, modularity and hierarchy. However, object-orientation as practiced over the past few years has produced mixed results, whereas Cleanroom has a significant track record of producting highly reliable systems, with extremely low after-delivery defect rates.

The study was based on the assumptions that 1) object-oriented methods support domain-specific architecture-based reuse, 2) Cleanroom software development emphasizes process-driven software development, and 3) object-oriented and Cleanroom ideas are both complementary and compatible.

The purpose of this paper is to 1) discuss why object-oriented and Cleanroom software engineering techniques should be integrated, 2) outline the generic process for object-oriented software development that was derived on STARS Task IA09, and comment on relevant aspects of the mapping from the studied methods to each generic process activity, and 3) discuss the shared leveraging of Cleanroom and object-oriented techniques, and how the integration of these techniques might be leveraged to produce software of greater reliability and reusability.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Cleanroom, Object-Oriented Techniques, Object-Oriented Methods, Object-Orientation, Software Engineering	24
	16. PRICE CODE: N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Title: Integrating Cleanroom with OO Methods for Reliable Software Development
Presenter: William H. Ett
Track: Track 5, Cleanroom
Day: Wednesday, April 24, 1996, 1030-1100
Keywords: Cleanroom, Certification, Object-Orientation, Object-Oriented Analysis, Object-Oriented Design, Object-Oriented Development, Method Integration
Abstract: Object-Orientation has shown much promise to support the building of systems from large and small grained components. However, application of OO methods have produced mixed results in terms of the quality of software delivered and its development costs. Cleanroom software engineering has a proven track record of the development of reliable systems, and with little exception, within cost and schedule constraints. The purpose of this paper is to discuss the need for integrating aspects of Cleanroom with object-oriented methods, and how, based on the analysis performed on STARS task IA09, the integration of both might be leveraged to produce software that is not only reusable, but of better quality and higher reliability, over currently practiced methods.

Integrating Cleanroom with OO Methods for Reliable Software Development

Introduction

STARS Task IA09, hereinafter referred to as "the study," was conceived to examine the potential complementary nature of the Cleanroom Engineering of software and a representative set of the popular object-oriented methods being used to specify, design, and develop software systems. Both methods of software development support the software concepts of abstraction, encapsulation, modularity, and hierarchy. Object-orientation as practiced over the past few years, however, has produced mixed results, whereas Cleanroom has a significant track record of producing highly reliable systems, with extremely low after-delivery defect rates.

The study was based on the assumptions that 1) object-oriented (OO) methods support domain-specific, architecture-based reuse, 2) Cleanroom software development emphasizes process-driven software development, and 3) object-oriented and Cleanroom ideas are both complementary and compatible [Ett95].

The purpose of this paper is to: 1) discuss why object-oriented and Cleanroom software engineering techniques should be integrated, 2) outline the generic process for object-oriented software development that was derived on STARS Task IA09, and comment on relevant aspects of the mapping from the studied methods to each generic process activity, and 3) discuss the shared leveraging of Cleanroom and object-oriented techniques, and how the integration of these techniques might be leveraged to produce software of greater reliability and reusability.

The paper is organized in three sections:

- **Object-Orientation and Cleanroom** - discusses what they are and why they are worth integrating,
- **Cleanroom and OO Method Integration Investigation** - discusses the approach followed on IA09 to analyze how the OO and Cleanroom methods might be integrated and presents commentary from the mapping of OO and Cleanroom method activities to the appropriate activities of the IA09-developed generic OO process, and
- **The Shared Leveraging of Cleanroom and OO Techniques** - discusses possible areas for technique integration, where both methods may benefit, and presents conclusions.

Object-Orientation and Cleanroom

Why Object-Oriented Development?

Object-oriented development is about analyzing, designing and implementing systems that comprise collaborating objects, where each object encapsulates the data and methods necessary to satisfy its processing requests. Object-orientation emphasizes the specification of the external interfaces of objects, and requires the practice of information hiding and the encapsulation of functions and data that perform the work of the object. Objects provide a convenient concept in which to think about the composition of systems. System composition through objects requires thinking about the architecture for a system in terms of assembling systems through the use of both large and fine grained components. Viewing systems as a composition of collaborating objects also supports the idea of developing not just reusable assets, but domain-specific solution architectures for classes of problems, where architectures, as well as objects, become units of system development and integration.

In practice, even though an enormous amount of software has been developed, only a portion of it is recoverable from "mining and refining" efforts. As software is developed in the future, and good software engineering techniques are employed to define and develop robust software objects, we may one day build up a sufficient quantity of software objects such that systems can be composed from reusable components, and software development may become more of an integration activity than one of development. One of the most important contributions of OO methods is the concept of developing reusable classes, which through inheritance, may be specialized. This specializing of generalized classes and their methods permits some methods to be inherited without modification, and allows others to be specialized as necessary to satisfy unique processing requirements not addressed by a parent class. It is the class concept that is the driving force behind the composition of systems from reusable components.

[Booch94] describes the underlying model (meta-model) upon which all object-oriented methods are based. He identifies the major elements in this meta model as the software techniques of abstraction, encapsulation, modularity, and hierarchy. The minor elements of the [Booch94] meta-model are typing, concurrency and persistence. These elements form the conceptual framework for the development of classes that represent the behaviors and properties of an object type and their integration into a system. [Booch94] also identifies three categories of methods that are used to support system analysis and design: top-down structured design, data-driven design, and object-oriented design. Top-down structured design is algorithmic decomposition. Data-driven design derives the structure of software systems by mapping system inputs to system outputs.

One of the most important characteristics of object-orientation is its focus on the behaviors of the system we are to develop, and the behavior of its objects. It is through this description of object behaviors, that we define the stimulus sequences and responses involved in a system communicating with external objects, and among communicating objects within the system.

The discipline called "object-oriented analysis" has recognized the importance of understanding the behavior of the software systems and objects of which they are to be composed. [Booch94], [Shlaer92], and [Jacobson92] all identify the need for performing analysis of and developing models to describe the behavior of a system. It is interesting to note that one of the most important aspects of Cleanroom is the development of an implementation- and state-free behavioral specification for a proposed system and each of its objects (black boxes).

Why Object-Orientation Alone Is Not Sufficient

Although object-orientation has evolved from work performed over three decades, it is still evolving as a discipline. The OO approach for specifying an object's behavior can best be characterized as heuristic. Rather than rigorously specifying the black-box behavior of a system and its objects, some OO techniques permit the discovery of object behaviors into the design process ([Booch94], p. 252), ([Jacobson], p. 157).

A system specification may only be considered complete when the behavior a system must exhibit and the transactions (scenarios) that support those behaviors have been described, and when all system behaviors, their transactions, and their supporting elements, have been verified against the requirements for a system. Most OO methods are not clear concerning the need for documented requirements, against which a system will be verified and validated. They have traded the concept of a specification for an "executable specification model" or "executable prototypes." This is not necessarily bad, but may be problematic since most OO methods choose to discover requirements

(new system behaviors) past the analysis phase. This an open invitation to requirements creep.

The most successful instances of OO method application have been through evolutionary software development. Evolutionary software development may provide excellent results, given there is sufficient schedule and budget to support that development. There are no effective measures to describe how good an object-oriented developed system is, although metrics have been identified to measure object-oriented development. The only real assurance one has that the system works is through coverage testing. Testing systems on the basis of crafted scenarios provides anecdotal evidence of system correctness. Testing might better be performed from models that describe the operational use of a system, and from which unbiased test scripts may be prepared. Further, testing of systems developed in object-oriented programming languages is often difficult [Perry90]. Strategies and techniques for testing software written using object-oriented languages is still a subject for continued research, development, and evaluation.

The object-oriented analysis and design methods that were studied (Booch, Objectory, Shlaer-Mellor) require the development of models that describe the *machine-independent logical design*, and the *implementation-dependent design* of a proposed system.

For the most part, there are no real strategies for verifying and validating these models to determine their completeness or consistency against stated requirements, with the exception of Objectory [Jacobson92]. Two out of three of the methods studied indicated that they expected requirements to grow and be refined, as the models were developed and reviewed with system stakeholders. When requirements grow, so do system behaviors. If system behaviors are not properly defined before development, none of its stakeholders may know what kind of system they will get, except through the continuous development of prototypes to test and confirm system behaviors with its stakeholders and other developers. The later these system behaviors are discovered in the development cycle, the more costly they are to address.

Finally, until OO methods provide us with answers to the following questions, OO methods alone are not sufficient to support systems development:

1. How much of the system's behavior do we need to understand before a system may be designed?
2. How much does it cost us to learn system behaviors and requirements as we design and develop systems and how can this learning be minimized?
3. Without a documented set of requirements for a system, how may we validate that the system specification or specification model satisfies the requirements for the system developed?

4. How may we certify the correctness of a system with respect to its proposed use, such that we are satisfied that the system addresses the needs of its users, on the basis of our knowledge of how each system user will use the system?

Why Cleanroom?

The Cleanroom engineering of software employs established engineering formalisms for the specification of software, that are founded in the mathematical concept of functions. This mathematical foundation provides software engineering with tools for verifying specifications and designs. A Cleanroom black-box specification is a complete behavioral description, where the black box is characterized by the transactions it supports, and each transaction is completely described through the analysis of the sequence of all stimulus/response pairs (transitions) formed to produce the transaction's result.

Cleanroom also emphasizes the concept of usage testing, where Markov models are developed to describe how the users of a system will exercise it. The resulting Markov model represents a usage model for the system and supports the concept of statistical testing. Because statistical testing is employed, Cleanroom can support software development under statistical quality control [Cobb90].

Software is thoroughly specified and then designed and implemented as a pipeline of small software increments. Each software object produced using Cleanroom requires its developers to: 1) specify the black-box behavior of every software object, 2) ensure there is sufficient and persistent data to ensure that the software object can support the transactions it must process, 3) ensure that the object encapsulates and maintains transaction history (stimulus history) required for the object to process new transactions, 4) demonstrate that the implementation of all behaviors are consistent with the black-box specification of the object, and 5) show that the developed system will produce the required results. Although this is not commonly understood and often mis-communicated, one of the chief principles of the Cleanroom engineering of software is to define and verify the correctness of system and system component behaviors.

Once a software increment is implemented, it must be certified against a model that describes the operational use of a system. Random test cases may be generated from this usage model to test each system increment, as well as the final system [Whittaker93]. This certification of each software increment and the final system defines the expected reliability for a system in terms of its mean time to failure, as well as other certification statistics [Whittaker93]. These statistics may be used as an indicator in analyzing the performance of the development process [Poore95]. These statistics may also be used to support a project's or organization's statistical process improvement initiative. Poor certification

statistics and testing failures usually indicate a problem that requires investigating, e.g., the process is not being properly followed, process adaptations are required, etc.

Another important aspect of Cleanroom is the development of verifiably correct software. This process begins with verifying that the system-level black box satisfies its stated requirements. It continues with the verification that design and implementation of the system satisfies all defined black-box behaviors. Cleanroom Software Engineering has a significant track record of successful application compared with other software engineering methods [Hausler94].

Cleanroom box structure development is based on the same conceptual foundation of every object-oriented method, i.e., abstraction, encapsulation, modularity, and hierarchy [Hevner93].

Why Integrate the Techniques of Both?

There are some excellent ideas that have evolved from the practice of object-orientation and of Cleanroom. OO methods provide valuable techniques for analyzing problem domains, both for a specific application and for a family of applications. These methods also provide techniques for the design and development of systems using reusable classes, where methods from a class may be inherited by another and specialized as required to satisfy processing needs. Cleanroom has produced proven techniques for (1) specifying the precise behavior of a proposed system (2) validating system specifications against their requirements, (3) verifying the correctness of the implementation of the external and each internal box structure design, (4) modeling the behavior of the system on the basis of the system's intended use, and (5) certifying the behavior of the software increments with respect to its intended use. Although there are areas of overlap between OO methods and Cleanroom, several of Cleanroom's strengths complement OO methods. If one considers the questions presented in the discussion of why OO alone in not sufficient as a systems development method from a Cleanroom perspective, the need for Cleanroom and OO integration becomes clear. Cleanroom has answers to these questions.

How much of the system's behavior do we need to understand before a system may be designed? The Cleanroom answer to this question is to iteratively develop a system-level black-box specification from existing and discovered system requirements, and other requirements sources, until the specification precisely describes the behavior that a proposed system must exhibit. Black box specification work is often supported by the development of prototypes to help system specifiers better understand and accurately define system and user requirements. One of the purposes of any analysis phase is to identify and synthesize the requirements for a system and engineer them into a precise behavioral specification or model. The goal of any specification is to be

complete, so that once design work begins in earnest, developers will discover few new system behaviors that the system must support.

How much does it cost us to learn system behaviors and requirements as we design and develop systems, and how can this learning be minimized? The Cleanroom answer to this question is to minimize the learning of system behaviors during development, by performing proper analysis up front. It has been shown that when missing requirements are discovered well into system design and development, they are more expensive to address. Will preparing a system-level black-box specification guarantee that missing requirements will not be found? Of course not - but, one can be almost certain that there will be fewer surprises during development with a system-level black-box specification than without.

Without a documented set of requirements for a system, how may we validate that the system specification or specification model satisfies the requirements for the system developed? The Cleanroom answer to this question is that we *must* prepare a system-level black-box specification that precisely describes the behavior the system is to exhibit. A system-level black-box specification must include a validation argument that describes the traceability of every system stimulus and response to a system requirement and its source.

How may we certify the correctness of a system with respect to its proposed use, such that we are satisfied that the system addresses the needs of its users, on the basis of our knowledge of how each system user will use the system? The Cleanroom answer to this question is to prepare a model that describes how the proposed system is intended to be used by all of its external users. The resulting usage model supports system certification and the estimation of the system's reliability with respect to the system's intended use. Certifying a system in this way directly addresses the issue of whether the system addresses the needs of its users, given that the usage model accurately represents how users will use the system.

Objectory and Booch described the importance of collecting measurements and data to support the periodic process improvement. All three methods could benefit from examining the Cleanroom process. The Cleanroom process defines protocols for the review of each process product. Cleanroom also requires that a periodic process review be performed on the application of Cleanroom techniques and their performance results. Cleanroom certification provides natural measurements of software product quality in terms of errors discovered and software reliability. Cleanroom process and certification ideas may be combined with OO process performance measures to define and instrument a process for performing each OO method.

Cleanroom and OO Method Integration Investigation

Study Approach

Many studies comparing the tools and techniques of the studied OO methods have been published. To provide a new dimension to this body of work, the focus of the study was placed on comparing the Booch, Objectory, Shlaer-Mellor, and Cleanroom methods from a process perspective.

To ensure that the documentation baseline that described each method to be studied was accurate, all three OO method authors were contacted. Each method author cited the books, technical reports and papers that accurately described their method. After examining the documentation baseline for each method, a composite of the phases, activities, and work products were drawn from to define a generic process to support the study. The generic process was used to examine the life cycle coverage of the selected methods. This composite view defined a fairly complete system development life cycle definition that was suitable for supporting the study's method analysis and mapping efforts.

After the documentation baseline was established and the generic process was defined, each method was described in terms of its phases, activities and work products. These method descriptions were used to support the mapping and analysis work of the study. This work resulted in the following artifacts, which are included in [Ett95]:

- Documentation of each method that includes a glossary, and a description of the phases, activities, and work products of the method.
- A mapping of the phases and activities of each studied OO method to Cleanroom. Commentary regarding potential Cleanroom integration with each method. An example page is included in Appendix A of the paper.
- A mapping of the phases and activities of each studied method to the generic OO process. Each mapping of a method's activities and work products was examined as possible candidate techniques that could be employed to tailor an instance of the generic OO process. Commentary on the mapping from a method's activity and work products to and support for a generic OO process activity was also prepared. An example page is included in Appendix B of the paper.

The commentary from [Ett95] associated with the mapping of OO and Cleanroom method activities identifies areas that should be closely examined, when considering integrating object-oriented methods and Cleanroom to prepare an integrated process. Each activity from the generic process is identified, along with the mapping discussion from [Ett95].

Commentary Regarding Method Applicability to Generic Process Activities

1. Concept Definition - Generic Process Activities and Commentary

1.1 Mission Statement	There is nothing object-oriented about concept definition. It is an important step, however, in establishing the context for systems analysis. Booch's "Vision of a Project's Requirements" and Objectory and Cleanroom early requirements statements all address concept definition.

2. System Analysis - Generic Process Activities and Commentary

2.1 Analyze Problem Domain	Booch, Objectory, and Shlaer-Mellor all have work products that describe the environment in which a system will operate and the domain objects of that environment. Objectory employs a particularly popular approach for describing "use cases" from which domain objects are identified.
2.2 Analyze Requirements	Objectory's Use-Case Model is thorough, formal, and in the customer's language. It may be supported by Booch micro-process activities for exploratory prototyping.
2.3 Plan Specification and Design Activities	Although all the methods address planning, Objectory provides the most comprehensive planning recommendations. Objectory relies on the completion of Use-Case analysis to scope the remaining analysis and design effort. A risk assessment should be incorporated in the planning process, as does Booch.
2.4 Review Analysis Phase Work Products	Booch, Objectory, and Shlaer-Mellor define criteria for examining their analysis products. Objectory's review criteria are the most comprehensive of the three processes for reviewing the results of a requirements analysis.

3. System Specification Generic Process Activities and Commentary

3.1 Specify User Interface	Early specification of the user interface is extremely important. Objectory addresses user interface design in requirements analysis, and Cleanroom does so in top-level black-box specification. Booch also suggests that executable prototypes be developed to demonstrate interface concepts to users.
3.2 Describe Usage Scenarios	Booch, Objectory, and Cleanroom call for the development of models that describe usage scenarios. The Cleanroom Usage Model is formalized as a Markov chain of usage states and transition probabilities between states. Either Objectory Use Cases or a Shlaer-Mellor system-level Object State Model could be used to prepare a Cleanroom Markov Usage Model. As a well-understood formalism, a Markov chain usage model can be analyzed to optimize development and testing resources and can be used as a test-case generator.
3.3 Specify Software System	The Objectory Use Case Model and the Cleanroom Box Structure Specification together are excellent (and complementary) approaches for specifying the external behavior of a system. An Objectory Use Case Model may be formalized in a Cleanroom black-box functional (mathematical) specification that maps sequences of external stimuli to external responses.
3.4 Review Specification Phase Work Products	Both Objectory and Cleanroom present criteria for reviewing the correctness, consistency, and completeness of specifications. Booch also identifies the importance of validating aspects of the specification for a system by conducting scenario walkthroughs.

4. System Design Generic Process Activities and Commentary

4.1 Identify Logical System Objects	All methods have activities for modeling logical software objects and their characteristics. Notable approaches to this activity are Objectory, which requires objects to be identified as to role (i.e., interface object, entity object, or control object), and Cleanroom, in which internal state objects are the explicit encapsulation of an object's external stimulus history.
4.2 Prepare Implementation-Independent Software Design	All methods have techniques for implementation-independent logical design. Cleanroom further enforces the mathematical properties of referential transparency and functional verifiability. Any of the methods would be reinforced by verification as a Cleanroom Box Structure Design.
4.3 Develop Software Architecture	All the methods provide for defining the software architecture. All have formalisms for depicting architectural structures and their relationships. Booch class diagrams, Objectory subsystem/package diagrams and Shlaer-Mellor Class Diagrams and Structure Charts appear equally useful for supporting architectural representations. Cleanroom architecture is represented as the top-level clear box at the highest level and the full box structure hierarchy in complete form. A Cleanroom architectural description has the merit of functional (mathematical) verifiability of designs to specifications throughout the hierarchy. Without further study and the comparison of actual examples, it is difficult to recommend any one of the approaches over others.

4. System Design Generic Process Activities and Commentary (Continued)

4.4 Specify Subsystems	A Cleanroom black-box specification completely specifies an object's external behavior. In an architecture of communicating objects, black-box specification of objects (in this case, subsystems) ensures the mathematical principle of referential transparency in system design. Referential Transparency - an example: Once an entity is defined as the number "8," it may be implemented as (6 + 2), (7 + 1), (3 + 1 + 4), or any other equivalent representation of the number "8" without regard to how the number "8" will be used.
4.5 Review of Design Phase Work Products	All of the methods provide some guidance for the review of the implementation-independent design and the system architecture. Of these, Objectory, Shlaer-Mellor, and Cleanroom have well -documented review criteria.

5. System Implementation Generic Process Activities and Mapping Commentary

5.1 Plan Increments	Booch, Objectory, and Cleanroom all address increment planning. Of these, the Objectory and Cleanroom methods are described most fully.
5.2 Develop Increment	Objectory and Cleanroom offer the most cohesive approaches to designing and implementing software increments. Shlaer-Mellor uses a "translation" approach, where generic mechanisms and structures in the architecture (referred to as "archetypes") are completed for the application. Overall, Cleanroom appears to minimize the risk of defects in implementation, and Shlaer-Mellor appears to maximize the potential for reuse.

5. System Implementation Generic Process Activities and Mapping Commentary (Continued)

5.3 Test Increment	Cleanroom is the only method to support statistical certification of software. Cleanroom certification views software testing as a statistical experiment, and yields a scientifically valid estimate of reliability.
5.4 Review Increment Work Products	Objectory and Cleanroom are the only processes to identify protocols and review criteria for reviewing the results of software increments.

The Shared Leveraging of Cleanroom and OO Techniques

Shared Fundamentals

Cleanroom software engineering and the studied OO methods are complementary. There is broad agreement that:

- objects are defined by their external behavior and their internal data and access programs;
- systems are defined by their external usage scenarios and heir internal organization of object accesses; and
- abstraction, decomposition, hierarchy, and other strategies are all important in identifying and relating the parts of a problem.

Furthermore, there is no more difference between a particular OO method and Cleanroom than there is between the particular OO methods studied -- in some cases the differences are less. There may be more difference between the Objectory and Shlaer-Mellor approaches, for example, than between Objectory and Cleanroom.

Difference in Focus

The OO and Cleanroom methods focus on different aspects of software quality. OO is generally focused on reusability, and Cleanroom is generally focused on reliability. Indeed, all OO and Cleanroom practitioners are concerned with both these aspects of software quality and more, but the aforementioned difference in focus is significant.

The study report, the "Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering" is not about resolving conflict between OO and Cleanroom, but about identifying the leverage that each can find in an explicit alliance with the other.

OO Leverage for Cleanroom

For those whose base is Cleanroom, the alliance is simple: add the OO analysis phase activities (in any of the methods in this Guide) to the Cleanroom process prior to Specification. The "thought models" in OO analysis aid in problem understanding and will set the stage for a rigorous Cleanroom specification.

Cleanroom Leverage for OO

For those whose base is an OO method, the alliance with Cleanroom has several points of leverage. The following aspects of Cleanroom are not typically found in OO processes, but could be included to support the studied methods:

- a system-level black-box specification
- a black-box specification of every object as a mathematical function
- system decomposition under the mathematical principle of referential transparency
- team verification that an object's implementation is a correct realization of the object's mathematical function specification
- use of usage models for statistical test case generation
- statistically valid reliability certification
- process review and improvement.

For more detailed discussion of these leverage points, see section III of [Ett95].

Potential Cleanroom Leverage for OO

The alliance with Cleanroom may have other points of leverage. But, these must be investigated further before their integration can be recommended. They are identified here because they were not examined or identified in the study report:

- use of Cleanroom certification techniques to certify design models
- use of Cleanroom certification techniques to independently certify software objects with the potential for widespread reuse, or access volume, i.e., a software component is frequently accessed by some or all of the components of a system
- analysis of defined objects for transaction closure.

Certifying Design Models. Although this would require further analysis and investigation, usage models constructed to support statistical test-case generation could be used both, (1) to support certification testing of software releases and (2) to certify an OO machine-independent logical design of a proposed system. Such testing could support certification of design models that represent the processing a system is to perform, such as a Shlaer-Mellor OO analysis model. This is especially important where these OO design models will be processed by CASE tools and translated into code.

Certifying Reusable or High-Use Software Objects. Usage models are developed to support the testing of a system on the basis of its intended use. All objects have users and exhibit black-box behavior. As a result, usage models could be independently prepared for software objects, and they could be independently certified. In the case where an object is to be reused, the accompanying usage model would support independent verification of the object's expected reliability.

Analyzing Objects for Transaction Closure. Regardless of whether box-structure techniques are employed to specify and design software objects, the Cleanroom concept of transaction closure is an important principle against which a software object could be analyzed. To examine a software object for transaction closure, one must determine first, whether the transactions the software object must process are sufficient to generate all required state data, and second, whether the state data within the object is sufficient to support all of the transactions. The second condition may be true only if the software object encapsulates the message (stimulus) history that is needed to process its transactions.

The Limits of Integration

Each of the methods addressed in [Ett95] has its own conceptual foundation:

- For Booch, "software growing" occurs through the iterative and opportunistic interplay of macro and micro processes in "round-trip Gestalt design."
- In Objectory, a set of "use cases" that define all system behaviors is elaborated to a fully traceable design that is implemented through staged incremental development.
- In Shlaer-Mellor, domain partitioning drives analysis activity, and domain-specific OO analysis models are translated to code using generic architectural components ("archetypes") [Shlaer93].

- In Cleanroom, engineering formalisms underlie incremental development; mathematical formalisms underlie specification, design, and correctness verification, and statistical formalisms underlie certification testing.

Integration (or any other form of combination) of the processes must occur with the same concern for conceptual integrity that must be observed in software product development.

Conclusions

A variety of approaches to integrating OO and Cleanroom are offered in [Ett95]. It was not a study task to remake or modify the methods developed by their distinguished authors. It was a study task to explore integration possibilities. From the work performed on the study, a conclusion about the viability of the approaches may be drawn. [Ett95] states: "Cleanroom is more like OO than OO is like Cleanroom, from the perspective of OO's underlying software fundamentals of abstraction, encapsulation, modularity, and hierarchy. It would be far easier to add an OO analysis to the front-end of a Cleanroom process than to insert the key Cleanroom characteristics (above) into an OO process."

Despite the complementary relationship between Cleanroom and OO methods, and processes derived from them, there are adoption barriers to this integration. These adoption barriers do not appear to be technical. But because Cleanroom engineering is new to many object-oriented method practitioners, compelling evidence must be provided that demonstrates that the integration of Cleanroom and OO methods will lead to highly reliable and reusable systems. These barriers include:

- the lack of a head-to-head comparison and analysis of the results from employing the studied OO methods and Cleanroom to develop *complete solutions* to a standard problem
- the lack of a formal analysis of the conceptual model upon which each OO method is based. It may only be through an understanding of the conceptual and mathematical basis of each OO method, that a determination can be made whether fundamental Cleanroom concepts may be practically integrated with the OO method, without re-engineering the method.
- the lack of support by tool vendors for Cleanroom software engineering techniques.

Addressing the first two items in the list may provide the justification for CASE tool vendors to consider providing support for Cleanroom Software Engineering. But, work needs to be performed to demonstrate to the OO community, on the basis of practical results, that such integration is both necessary and practical. Where Cleanroom techniques may be easily added to the stated OO methods,

technique integration may depend on successful demonstration, cost-benefit analysis, and the desire and will to improve software quality.

IA09 Report Availability

The final STARS Task IA09 Report, "A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering," will be available for review on the World Wide Web, on March 1, 1996. The URL for the report will be:

http://source.asset.com/stars/loral/cleanroom/oo/guide.html

The reader also may wish to review the STARS Cleanroom tutorial, also available for review on the World Wide Web. The URL for this tutorial is:

http://source.asset.com/stars/loral/cleanroom/tutorial/cleanroom.html

References

[Booch94] Booch, Grady. Object-Oriented Analysis and Design with Applications. Benjamin-Cummings. 1994.

[Cobb90] Cobb, Richard, and Harlan Mills. "Engineering Software under Statistical Quality Control." IEEE Software, November 1990.

[Cosmo94] Cosmo, Henrick. Black-box specification Language for Software Systems. Masters Thesis. Department of Communication Systems. Lund University, Sweden. 1994.

[Ett95] Ett, William and Carmen Trammell. A Guide to Integration of Object-Oriented Methods and Cleanroom Software Engineering. STARS Task Final - Comment Draft. Loral Federal Systems, December 22, 1995.

[Hausler94] Hausler, Philip, Richard Linger, and Carmen Trammell. "Adopting Cleanroom Software Engineering with a Phased Approach." IBM Systems Journal, Volume 33, Number 1, 1994.

[Hevner93] Hevner, Alan and Harlan Mills. "Box-Structured Development Method for System Development with Objects." IBM Systems Journal. Volume 32, Number 2, 1993.

[Jacobson92] Jacobson, Ivar, Magnus Christerson, M. Jonsson, and G. Overgaard., Object-Oriented Software Engineering. Addison-Wesley, 1992.

[Mills86] Mills, Harlan, Richard Linger and Alan Hevner. Principles of Information Systems Analysis and Design. Academic Press. 1986.

[Perry90] Perry, Dewayne and Gail Kaiser, "Adequate Testing and Object-Oriented Programming." Journal of Object-Oriented Programming, Volume 2, Number 5, January-February, 1990.

[Poore95] Poore, Jesse. "Usage Testing as Engineering Practice." European International Symposium on Cleanroom Software Engineering. Berlin, Germany, March 28-29, 1995.

[Shlaer92] Shlaer, Sally and Steve Mellor Object-Oriented Lifecycles: Modeling the World in States. Prentice-Hall. 1992.

[Shlaer93] Shlaer, Sally and Steve. Mellor, "The Shlaer-Mellor Method," Project Technology, Inc., Technical Report, 1993.

[Whittaker93] Whittaker, James and Jesse Poore. "Markov Analysis of Software Specifications." ACM Transactions on Software Engineering and Methodology, Volume 2, Number 1, January 1993.

Acknowledgments

This paper is based in part from work performed on the ARPA STARS Task IA09: "Integration of Cleanroom into Object-oriented Specification and Design Methods," performed by Loral Federal Systems, Software Engineering Technology, Incorporated and the University of Tennessee's Software Quality Research Laboratory.

The author wishes to express his appreciation to Linda Brown, ARPA STARS Program Manager, and the Department of Defense's C3I Office for their support of this work. The author also wishes to express his appreciation to Dr. Carmen Trammell for her collaboration in the IA09 study. Finally, the author wishes to thank Dr. Trammell and STARS Program Manager Dave Ceely for their excellent comments on earlier drafts of this paper.

Biography

William H. Ett

William H. Ett is an advisory programmer for Loral Federal Systems, assigned to the ARPA STARS Program. During STARS, Mr. Ett has been actively involved in process and SEE integration, and helped transfer STARS process technology to the STARS/Air Force Demonstration Project. Mr. Ett also was involved in several Cleanroom related tasks on STARS. He was the principal architect for the STARS Cleanroom Engineering Process Assistant and was the Loral technical lead for the STARS Cleanroom Object-Orientation Integration Study.

Mr. Ett has held systems and software engineering positions in the Government and private sectors, including the Navy Bureau of Medicine and Surgery, Morrison-Knudsen, MA/COM, and IBM Federal Systems Company. Mr. Ett's work experiences include the management, design and development of information systems, development of knowledge-based systems and tools to facilitate knowledge acquisition, and the design, development and integration of environments to support software process management.

He holds a B.S. in Computer Science and Information Systems from the University of Maryland and has done graduate work at George Washington University in Operations Research. Mr. Ett is a member of the ACM and IEEE.

William H. Ett
Loral Federal Systems
700 North Frederick Avenue
Gaithersburg, MD 20879
Voice: (301) 240-6322
Fax: (301) 240-6073
Internet: ettb@lfs.loral.com.

Appendix A: Cleanroom/Method Integration Discussion Sample

Included is a sample page from [Ett95], illustrating the mapping of Objectory activities to Cleanroom activities, and the commentary regarding Cleanroom-Objectory integration.

CLEANROOM EXTENSIONS TO THE OBJECTORY PROCESS

The Objectory process is given in the left column as the "base" process, and the Cleanroom process in the right column is mapped to the Objectory process. Comments that straddle the columns are recommended Cleanroom extensions to Objectory. Cleanroom extensions are given immediately after the Objectory activity to which they apply.

OBJECTORY PROCESS	CLEANROOM PROCESS
1. Project Planning	**1. Project Management**
1.1 Objectory Process Configuration *Development Case Description*	*Process References*
1.2 Objectory/Project Management Integration *Tailored Project Management Process*	
1.3 Objectory/Configuration Management Integration *Tailored Configuration Management Process*	
1.4 Project Scoping	
1.5 Project Organization *Project Technical Staffing Requirements*	**1.5 Tasking** *Tasking Records*
1.6 Project Planning *Project Plan*	**1.3 Schedule Development and Maintenance** *Project Schedule*
1.7 Development Case Installation *Trained Personnel*	**1.4 Training** **1.1 Customer Interaction** *Customer Requirements* **1.2 Process Control** *Process Control Standards*
Cleanroom extension: Include Cleanroom Process Control ideas in project management process. Rationale: Objectory has comprehensive instructions for tailoring the process for a software project. Provisions for process control would be a worthwhile addition.	

Appendix B: Generic OO/Cleanroom Integration Discussion Sample

Included is a sample page from [Ett95], illustrating the matrix prepared to discuss the coverage provided by each of the three methods and Cleanroom to support a generic process activity and showing where one or more of the selected methods are the most suitable candidates for preparing an integrated generic object-oriented process.

GENERIC OBJECT-ORIENTED CLEANROOM PROCESS

The Generic Process is given in the left column, and the other processes in the rightmost four columns are mapped to the Generic Process. The shaded activities in the Generic Process column represent the work to be done, and the shaded work products in the other columns represent options for performing the work. Comments that straddle the columns provide perspective on the options.

GENERIC PROCESS	BOOCH	OBJECTORY	SHLAER/MELLOR	CLEANROOM
1. Concept Definition	**1. Conceptualization**	Prestudy; Feasibility study **2.1 Requirements Analysis** ([3], p. 443)	-	
1.1 Define the Mission	Micro Process cycle to implement concept prototype	Develop & evaluate project needs and ideas; ([3], p. 444)	-	**1.1 Customer Interaction**
Mission Statement	*Executable Prototype* *Vision of the Project's Requirements*	*High level requirements* *Needs statement; ([3], p. 444)*	-	*Customer Requirements (initial)*
There is nothing object-oriented about concept definition. It is an important step, however, in establishing the context for systems analysis. Booch's "Vision of a Project's Requirements," and Objectory and Cleanroom early requirements statements all address concept definition.				
2. Systems Analysis	**2. Analysis**	**2. Analysis**	**1. Analysis**	
2.1 Analyze Problem Domain	**2.1 Domain Analysis**	**2.1 Requirements Analysis**	**1.1 Partition the System into Domains** **1.2.1 Build Object Information Model**	2.1.5 Top-Level Usage Specification Development